A Case Study for The Methodology for Resolution Mapping for Cross Resolution Simulation Applications using Event-B

Ahmet Kara1, 2
ahmet.kara@tubitak.gov.tr

Halit Oğuztüzün1
oguztuzn@ceng.metu.edu.tr

M. Nedim Alpdemir2
nedim.alpdemir@tubitak.gov.tr

1Department of Computer Engineering
Middle East Technical University
Ankara, Turkey

2TUBITAK BILGEM ILTAREN
06800 Umitkoy
Ankara, Turkey

Abstract
This paper proposes a case study for demonstrating “A Methodology for Resolution Mapping for Cross Resolution Simulation Applications using Event-B” manuscript.

1. CASE STUDY
To demonstrate our approach, we can think of a simple DEVS simulation application that is built by composing two models, a display model and a platform model (Figure 1). The Platform model calculates the values for various attributes of a Car object at each step of the simulation and produces this Car object as its output. The display model reads the Car object as its input and displays properties and position of a car on a graphical user interface.

Assume, for instance, that the developer of the physical model improves his/her model by introducing additional attributes such as calculating fuel level with more detailed variables, using vectoral position, direction and scalar velocity, to be able to operate at higher resolutions. As the model now has a higher resolution, the output entity has changed to DetailedCar object which contains more detailed attributes, and hence the input/output compatibility with the display model is broken (Figure 2). However, the graphical representation of a car does not actually require this high resolution and if we can reduce the output resolution of the detailed physical model we can continue to execute our simulation without any modification of the display model.

1.1. Using Event-B for Model Data Type Definitions
The Car object mentioned earlier can be represented with an Event-B machine as given in Figure 4. Notice that VARI-

ABLES section of the machine definition includes the attributes of the complex type ”car”. Note also that the INVARIANTs section contains the types the attributes (of some selected ones for convenience). In fact, in an Event-B machine, invariants has two major responsibilities:

- Data Type Specification: such as integer, string or even array and complex types.
- Provision of Semantic Information: The relations with other variables are given. For example: \( x > y \), range restrictions \( x < 1000 \) or more complex constraints \( \text{CurrentSpeed} \leq \text{Weight/EnginePower} * 100 \)

We further illustrate the use of invariants for the provision of additional semantic information on our Car object in Figure 5.

1.2. Applying Refinement
An example illustrating the use of glue-invariants for the purpose of defining a refinement from a Car entity to a DetailedCar entity (depicted in Figure 6) is given in Figure 7. Note that the DetailedCar machine contains more variables, some of its variables have complex data types (such Position having Vector3 type) and that it indicates, via REFINES key
inv10 : FuelState ≤ 100
inv11 : MaxSpeed * MaxSpeed ≥ velX * velX + velY * velY + velZ * velZ

Figure 5. More Invariants for the Event-B Machine of Car object

MACHINE Car
VARIABLES
    MaxSpeed
    Range
    FuelState
    posX
    posY
    posZ
    velX
    velY
    velZ
ININVARIANTS
    inv1 : MaxSpeed ∈ N
    inv4 : posX ∈ Z
    inv9 : velZ ∈ Z
END

Figure 4. Event-B Machine for Car object

word, that it is a refinement for the Car machine. Note also that the glue invariants define how aggregated variables of the Car entity can be obtained from the variables of DetailedCar entity. As a relatively subtle point, notice the use of accessor functions such as (Vector3.X) for variables of complex types in the expressions of glue-invariants. These functions give details for accessing the members of complex data types (X member of a Vector3 variable) and used for both proof system and code generation described on following sections.

1.3. Proof of Glue Invariants

In our DetailedCar model, as we used the TankCapacity variable in our glue-invariant glue3 in Figure 7 Rodin produced a Proof Obligation stating that TankCapacity = 0 would be an invalid state. This was because in glue3 there would be a division by zero error. So we defined a new invariant inv13 in Figure 6 to prevent this error.

1.4. Converter Generation

As such, the generation of a converter from DetailedCar entity to Car entity, for instance, fills the gap between our display model and new platform model and allows the simulation to execute as expected (Figure 3).

MACHINE DetailedCar
REFINES Car
SEES BaseModel, Car2Model
VARIABLES
    EnginePower
    Weight
    TankCapacity
    CurrentFuel
    FuelConsumption
    Position
    Direction
    CurrentSpeed
ININVARIANTS
    inv1 : EnginePower ∈ N
    inv2 : TankCapacity ∈ N
    inv7 : Position ∈ Vector3
    ...
    inv11 : CurrentSpeed ≤ Weight/EnginePower * 100
    inv13 : TankCapacity ≠ 0
END

Figure 6. Event-B Machine for DetailedCar object

glue2 : Range = TankCapacity/FuelConsumption
glue3 : FuelState = CurrentFuel/TankCapacity * 100
glue5 : posX = Vector3.X(Position)
glue6 : posY = Vector3.Y(Position)

Figure 7. Glue Invariants for DetailedCar object